Immersed Boundary Methods for Fluid-Structure Interaction and Shape Optimization within an FEM-based PDE Toolbox

Janos Benk, Hans-Joachim Bungartz, Miriam Mehl, and Michael Ulbrich

Abstract One of the main challenges in a classical mesh-based FEM-approach is the representation of complex geometries. This challenge is often tackled by a computationally costly mesh generation process, where the resulting mesh's facets represent the boundary. An alternative approach, that we employ here, is the immersed boundary (IB) approach. This uses instead a computationally cheaper structured adaptive Cartesian mesh and an explicit boundary representation, where the challenge mainly lies in the boundary condition (BC) imposition on the mesh cells intersected by the geometry's boundary. One IB method is Nitsche's method that we employ here for fluid-structure interaction (FSI) and shape optimization problems. The simulation of such complex physical systems modeled by PDEs requires a combination of sophisticated numerical methods. Implementing a FEM-based simulation software that computes a particular PDE's solution often requires the reusage of existing methods. In order to make our approach public and also to prove the modularity of it, we integrated our IB methods in an existing FEM-based PDE toolbox of the Trilinos project, called Sundance.

Janos Benk
Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
e-mail: benk@in.tum.de

Hans-Joachim Bungartz
Computer Science, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany
e-mail: bungartz@in.tum.de

Miriam Mehl
Institute for Advanced Study, Technische Universität München, Lichtenbergstr. 2a, 85748 Garching, Germany e-mail: mehl@in.tum.de

Michael Ulbrich
Mathematics, Technische Universität München, Boltzmannstr. 3, 85748 Garching, Germany e-mail: mulbrich@ma.tum.de

1 Introduction

The idea behind the development of a finite element PDE toolbox suitable not only for simulation but also optimization algorithms is to provide a software allowing for the reuse of mesh functionality, data processing, matrix assembly, linear and non-linear solvers for various kinds of PDE based applications. To establish a new application code, the user only has to provide a suitable formulation of the continuous equations and of the finite element basis. Sundance [27, 28] has been developed with exactly that task. The user implements the particular application in a weak form formulation. This makes also the implementation of dual equations for optimization very convenient. In addition to the problem formulation, the type of finite element basis to be used has to be prescribed – of course in close connection with the computational grid and, therewith, the element type. Element quadrature, matrix assembly, and (non-)linear solvers are given by Sundance or Trilinos [20], respectively.

The focus of this paper is on the numerical simulation of fluid-structure interaction processes and related optimization problems. These problems require the flexible handling of complex and moving geometries including large geometrical or even topological changes. In particular for optimization, a high computational and memory efficiency is additionally required. Based on these requirements, we added a further grid type to the Sundance toolbox: structured adaptively refined Cartesian grids. Due to its strict structuredness, this grid type makes the storage of any explicit structure information such as relations between vertices, edges, faces, and elements or between elements and their neighbours obsolete and, thus, reduces the storage requirements to a minimum.

However, a major drawback of Cartesian grids is their inherently poor approximation accuracy for complex geometries. Thus, sophisticated methods to improve this accuracy are required. Here, immersed boundary methods such as Nitsche's method are a promising approach. In this paper, we present a Cartesian grid implementation in Sundance including an extension of Nitsche's method to flow simulations on moving geometries, in particular fluid-structure interaction and shape optimization problems.

In the following section, we show the underlying equations of our fluid-structure scenarios and recall some basic methods for PDE constraint optimization. Sect. 3 gives a an overview of immersed boundary methods and the related terminology. In Sect. 4, we shortly introduce the Sundance toolbox with its basic functionality, user interface, and implementational concept. We explain the Cartesian grid implementation in Sect. 5 and introduce Nitsche's method for fluid dynamics in moving geometries (Sect. 6). Section 7 presents details on the implementation in Sundance, whereas numerical examples for fluid-structure interactions using Nitsche's method and their results are shown in Sect. 8. For shape optimization, we present first simple scenarios with results in Sect. 9. Finally, we summarize results and give an outlook on future work in the conclusion (Sect. 10).

2 The Fluid-Structure Interaction Model and Optimization Methods

The Fluid-Structure Interaction Model. For the simulation of fluid-structure interactions on Cartesian grids, we use the incompressible laminar Navier-Stokes equations

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \Delta \mathbf{v} - (\mathbf{v} \cdot \nabla)\mathbf{v} - \nabla p + \mathbf{f}_f \text{ (momentum equation)}, \tag{1}$$

$$\nabla \cdot \mathbf{v} = 0 \text{ (continuity equation)} \tag{2}$$

to model the fluid flow in connection with the structural dynamics equations

$$\rho_s \frac{\partial^2 \mathbf{u}}{\partial t^2} + \nabla \cdot \sigma_s + \mathbf{f}_s = 0, \text{ (structural dynamics)} \tag{3}$$

where $\mathbf{v}$ denotes the flow velocities, p the fluid pressure, $\mathbf{f}_f$ external forces acting on the fluid such as gravity forces, and ν the viscosity of the considered fluid. Further, $\mathbf{u}$ is the structure displacement vector, ρ_S the structure density, σ_s the stress tensor, and $\mathbf{f}_s$ denotes external traction forces exerted on the structure.

The interaction between fluid and structure is given by the continuity of velocities and the balance of forces

$$\mathbf{v} = \frac{\partial \mathbf{u}}{\partial t}, \tag{4}$$

$$\sigma_f \mathbf{n}_f = \sigma_s \mathbf{n}_s, \tag{5}$$

where σ_f and σ_s are the stress tensors of fluid and structure and $\mathbf{n}_f$ and $\mathbf{n}_s$ denote the normal vectors of the fluid and structure domain boundaries.

The structure is usually simulated in a Lagrangian framework, i.e., the grid deforms with the structure movement, such that the surface of the structure is always represented accurately. For the fluid domain, we, however, use an Eulerian fixed grid setting to allow also for large geometry or even topology changes.

Shape Optimization Methods. In our example applications, we consider designing the shape of a body B exposed to Navier-Stokes flow such that a given cost functional is minimized. We denote this functional with $J(y, \Omega)$ (e.g., the drag) under the constraints $E_\Omega(y) = 0$ (Navier-Stokes equations including boundary conditions), where $y = (\mathbf{v}, p)$ denotes the flow variables and Ω is the fluid domain surrounding the body to be optimized. Additional constraints such as constant volume of the body shape or smoothness of the body's boundary ensure the existence of an optimal design. A particular challenge of shape optimization is the fact that during the iterative process of optimization a sequence of different domains Ω_k is generated and the PDE has to be solved on each of these changing domains. This is closely related to boundary movement due to structure deformation in the fluid-structure interaction applications described above.

For gradient-based optimization, the derivative of $j(\Omega) := J(y(\Omega), \Omega)$ with respect to domain variations $\Omega \mapsto (\mathrm{Id} + V)(\Omega)$ is required, where V is a displacement field and Id the identity operator.

A classical approach, the method of mappings [6, 18, 33], works with a reference domain Ω_{ref} and bi-Lipschitz transformations $T : \mathbb{R}^n \to \mathbb{R}^n$ to represent the current domain via T as $\Omega = T(\Omega_{\mathrm{ref}})$. Using this transformation, the problem can be transported to the fixed domain Ω_{ref}. This results in a nonlinear PDE constrained optimal control problem with T serving as the control (design parameterization). If preferred, it can be arranged that the k-th optimization iteration performs its computations on the current domain Ω_k by viewing it as the reference domain. For conventional FEM, this requires moving the computational mesh as well as remeshing if the moved mesh deteriorates.

A closely related alternative to the method of mappings is provided by shape differential calculus [42, 47]. It offers a rich machinery for computing the directional derivative $dj(\Omega)[V] := \frac{d}{dt} j((\mathrm{Id} + tV)(\Omega))\big|_{t=0}$, called shape derivative if it exists in the Gâteaux sense, in the displacement direction V. The Hadamard-Zolesio structure theorem [42, 47] states that under suitable conditions there exists a distribution g on the boundary of Ω such that $dj(\Omega)[V] = \langle g, V \cdot n \rangle_{\partial\Omega}$. The distribution g represents the sensitivity of $j(\Omega)$ with respect to normal boundary variations. It can be used to develop shape optimization algorithms that move the boundary of Ω to achieve a descent method.

The shape gradient g or a discrete version of it can be obtained by the adjoint approach [21]. In contrast to the sensitivity method, the computational complexity of the adjoint approach is independent of the dimension of the considered space of boundary displacement directions V. It requires to compute the adjoint state. If we denote by $\mathbf{u}$ the state and abbreviate the PDE as $E(\mathbf{u}, \Omega) = 0$, then the adjoint state $\mathbf{w}$ solves the adjoint PDE

$$(E_{\mathbf{u}}(\mathbf{u}, \Omega)\mathbf{v}, \mathbf{w}) = -(J_{\mathbf{u}}(\mathbf{u}, \Omega), \mathbf{v}) \quad \forall\, \mathbf{v} \in U_0, \tag{6}$$

where U_0 is the space of state variations and the subscript denotes the partial Fréchet derivative of the respective operator or function.

3 Immersed Boundary Methods – A Short Overview

The term immersed boundary methods summarizes a class of methods aiming at the accurate description of complex and moving geometries in a fixed Cartesian grid. It was developed already in the seventies by Peskin [37] for the simulation of blood flow in the human heart. The basic idea is to embed a moving structure in a fluid flow simulated on a fixed (Eulerian) Cartesian grid. The influence of the structure on the flow is taken into account by a force function (which is zero away from the structure). A good overview on immersed boundary methods is given in [30].

In literature, there exists a variety of methods and a lot of names besides immersed boundaries that are related to the same topic: handling complex and moving geometries in fixed and mostly Cartesian grids. We shortly mention and classify the most important ones:

Marker-and-Cell. Early marker-and-cell methods, introduced by Harlow and Welch already in the sixties [19]) use cell-markes to defined fluid and non-fluid cells in a free surface flow. This approach obviously can easily be extended to complex flow geometries with obstacles [16] and to fluid-structure interaction [11]. However, the accuracy is restricted by the character of the grid, i.e., for Cartesian grids to $O(h)$ if h is the mesh width.

Volume-of-Fluid. To increase accuracy, volume-of-fluid methods have been introduced by Noh and Woodward in 1976 [35]. Instead of simply assigning cells the values zero for an empty cell and one for a cell completely filled with fluid, the volume-of-fluid method introduces a fractial function giving the fraction (in terms of volume) of a grid cell which is filled with fluid. This method has the advantage to allow a fulfillment of mass conservation. The accuracy of the actual geometry representation is only first order for the original method, but can be enhanced to second order (see, e.g., [38]).

Level-Set. To overcome the drawbacks of the volume-of-fluid method, another twelve years later, Osher and Sethian introduced the level-set method that tracks the fluid surface using a function that is zero at the domain boundary (and positive inside the flow domain) [36]. This allows even for topology changes in a natural way, conserves mass if applied correctly, and allows for high order of accuracy. The transient computation of the level-set function requires the solution of a Hamilton-Jacobi equation, which is neither trivial nor computationally cheap. The fractial function of the volume-of-fluid method can be interpreted as a volume integral over the level-set function.

Cut Cell. In contrast to the aforementioned approaches, cut cell methods [26] work with an explicit representation of the domain boundary by, e.g., polygons in 2D, triangulations in 3D, or higher-order surface representation with splines, for example. Classical cut cell methods use a finite volume discretization on the resulting geometry consisting of 'standard volumes' and 'cut cell volumes'. Accordingly, the roots of cut cell methods come from flow simulation. The accuracy depends on the accuracy of the boundary representation. A drawbacks of the cut cell method in particular in 3D is the large variety of possible types of cuts through a cell, and, therewith, the large number of different cut cell types. The advtantage is the easy geometry representation with straightforward possibilities to interface with CAD (Computer Aided Design), e.g.

Fictitious Domain. Also fictitious domain methods [10] have been developed originally for fluid dynamics but are applied as well for structural dynamics [39]. The complex geometry is embedded in a simpler domain such as a square in 2D or a cube in 3D, and the original equation is replaced by an equation on this simpler domain by introducing jumping coefficients. These jumping coefficients can be

interpreted as an extremely high or low stiffness outside the actual domain if the domain is surrounded by a rigid body or vacuum, e.g.. The resulting system can be discretized either by finite differences, finite volumes, or finite elements. The drawback of the fictitious domain method in this simple form is that it allows only the modeling of rather simple boundary conditions such as homogeneous Dirichlet or Neumann boundary conditions. As soon as other boundary conditions such as those originating from a constant movement of a rigid body in a fluid have to be applied, additional efforts are required such as adding a Lagrange multiplier to enforce the boundary condition (see, e.g., [15]). The accuracy is limited by the accuracy of the discrete representation of the location of coeffcient jumps as well as the influence of averaging effects when discretizing the underlying PDE at such a jump.

Penalty Methods. The penalty method proposed by Babuška in the late sixties and early seventies [1, 2] for the Poisson equation with homogeneous boundary conditions allows to weakly enforce boundary conditions with a finite element basis that is not conforming with these boundary conditions. It enhances the energy functional to be minimized by the finite element solution by a boundary penalty term $h^{-s}\int_{\partial\Omega} v^s dS$. In case of a partial differential equation $L(u) = f$ with a differential operator L, this gives:

$$F(v) = \int_{\Omega} L(v)v - 2fvd\Omega + h^{-s}\int_{\partial\Omega} v^s dS. \tag{7}$$

This methods performs well in practice. One can be shown to be sensitive with respect to the choice of s and to deteriorate the order of convergence of the original discretization in theory.

Nitsche's Method. Nitsche developed a closely related method [34], that also enhances the energy functional in a finite element setting by a boundary penalty term, but adds further terms ensuring the symmetry of the resulting discrete equations and, therewith ensures full consistency. Nitsche's method has been applied to computational fluid dynamics in [4] and [3]. In the latter, the method is called *weakly enforced boundary conditions*.

Extended Finite Elements. In contrast to penalty and Nitsche's methods, extended finite element methods [31] enhance the finite element basis (enriched basis functions) at sharp *internal* boundaries, where discontinuities in material properties or cracks cause discontinuities in the numerical solution of a partial differential equation.

In this paper, we describe an extension of Nitsche's method for transient problems with moving boundaries (Sect. 6). This is new and requires some additional techniques to ensure the numerical stability of the time-stepping scheme.

4 The Sundance Toolbox in a Nutshell

The Sundance toolbox is a convenient software providing the user with a vast functionality for several pieces of the so-called simulation pipeline: problem formulation, i.e., mesh generation and weak form formulation of the required partial differential equations, discretization including assembling of a finite element matrix, computation of element matrices and stiffness matrix assembly, and, finally, solvers for the resulting systems of linear or nonlinear discrete equations. Figure 1 shows the steps of the simulation pipeline and outlines the contributions from users, Sundance components and external software components are linked to Sundance via well-defined interfaces.

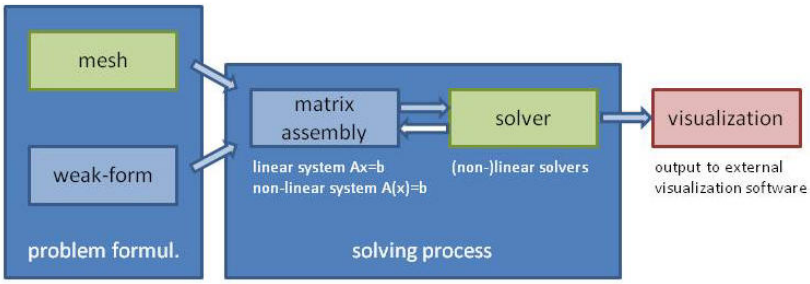

Fig. 1 Steps of the simulation pipeline in Sundance. Functionalities provided by Sundance are marked light blue, functionalities provided both by Sundance and by interfaces to external tools are green, and functionalities provided exclusively by interfaces to external software are marked red.

A computational grid in Sundance is considered as a collection of mesh entities (elements, faces, edges, vertices) and their relations. A mesh compatible with the Sundance interface has to provide the following functionalities:

1. return the unique identification number and dimensionality (0 for vertices, 1 for edges, 2 for faces, 3 for 3D elements) of a mesh entity,
2. return the position, i.e., the spatial coordinates of a given vertex,
3. return all lower-dimensional components of a mesh entity, e.g., all faces, edges, and vertices of a three- dimensional element,
4. return the indices of all mesh elements that contain a given mesh entity together with its indices within the respective elements.

The grid can either be generated by an external mesh generator and read via a Sundance compatible parser from an input file or be generated internally within Sundance given the characteristic parameters of the mesh. Whereas the first approach is very well suited for unstructured grids and assures a high flexibility with respect to geometry and mesh generation with existing user-trusted tools, the latter

is more suited for grids with a fixed structure such as our Cartesian grids. The mesh functionality described above can be either implemented by getting the required information from stored data (as for unstructured grids) or deriving information from the given structure of the grid.

For parallel simulations, the mesh interface is enhanced by functions tracing the assignment of mesh entities to processes:

1. return the ID of the owner process of a mesh entity,
2. transform the local ID of a mesh entity to the local ID on the respective process (and vice versa).

The domain partitioning itself can again be done via an interface to external established mesh partitioners or be implemented in Sundance itself.

As mentioned above, the equations describing the application are formulated by the user in a weak-form syntax. We give an example from [7] to demonstrate how easy a new application can be implemented in Sundance:

- We want to solve the two-dimensional Poisson equation, which is given by the weak form
$$\int_{\Omega} (\nabla \mathbf{u} \nabla \mathbf{v} - \mathbf{f}\mathbf{v})\, d\Omega = 0 \text{ for all } \mathbf{v} \text{ in } V.$$
- We start with reading a mesh that has been generated and written to a file by an external mesh generator:

```
MeshType meshType = new BasicSimplicialMeshType();
MeshSource meshReader =
  new TriangleMeshReader("meshInputFile.1", meshType);
Mesh meshExternal = meshReader.getMesh();
```

- Subsequently, we formulate the equations to be solved:

```
CellFilter Omega = new MaximalCellFilter();
CellFilter Boundary = new BoundaryCellFilter();

Expr unknBase = new Lagrange(1);
Expr testBase = new Lagrange(1);
Expr u = new UnknownFunction( unknBase , "u");
Expr v = new TestFunction( testBase , "v");
Expr dx = new Derivative(0);
Expr dy = new Derivative(1);
Expr grad = List(dx, dy);
QuadratureFamily quad = new GaussianQuadrature(2);
Expr weakForm =
  Integral( Omega , (grad*u)*(grad*v) - f*v, quad);
Expr bc = EssentialBC( Boundary , v*(u-1.0), quad);
```

- Finally, we setup the discrete problem and solve it with a suitable solver, which we choose from Trilinos [20]:

```
LinearProblem prob(mesh, weakForm, bc, v, u, vecType);

ParameterXMLFileReader reader("bicgstab.xml");
ParameterList solverParams = reader.getParameters();
LinearSolver<double> solver =
  LinearSolverBuilder::createSolver(solverParams);

Expr up = prob.solve(solver);
```

The example above shows a simple stationary linear example. For time-dependent applications, the time-stepping has to be defined by the user, in addition. Similarly, optimization problems require the formulation of the dual equations and the optimization algorithm by the user. Sundance does not implement an own visualization functionality but offers output functions writing the results in several standard formats suitable for different standard softwares such as VTK [41], ExodusII [25], and Matlab [29].

5 Cartesian Grids and Their Implementation in Sundance

Introduction of and General Remarks on Cartesian Grids In contrast to unstructured grids, tree-structured adaptive Cartesian grids have the advantage of being highly efficient in terms of storage requirements. Whereas for an unstructured grid all elements, faces, edges, nodes, and, in particular, their relations have to be stored explicitly, our Cartesian grids are defined by the chosen tree-structure such that the only information that has to be stored is whether a grid cell is further refined or not. The grid on the left in Fig. 2 is for example given by the bit sequence

$$1\ 0\ 1\ 0000\ 0\ 0,$$

where the first '1' denotes the root, i.e., the whole square domain, the subsequent '0' stands for the non-refined lower left cell at refinement level one, the '1' for the refined lower right cell, followed by four '0's for its children and two '0's for the non-refined upper left and upper right level one cells. Thus, in this example, the bitsequence is ordered in a depth-first manner with Morton order of the cells at each level. It shows that an arbitrary tree-structured adaptive Cartesian grid can be stored with a memory requirement of only one bit per grid cell (over all levels) if three important parameters defining the type of grid and the traversal order are given:

1. the refinement factor (in our example two per coordinate direction),
2. the type of tree-traversal (depth-first or breadth-first),
3. the traversal order of cells at the same level (in our example Morton order).

Similar efficiency arguments hold for the generation of an adaptive Cartesian grid in an arbitrary domain: Starting from a square or cube containing the whole computational domain, the grid is locally further refined wherever a refinement is required

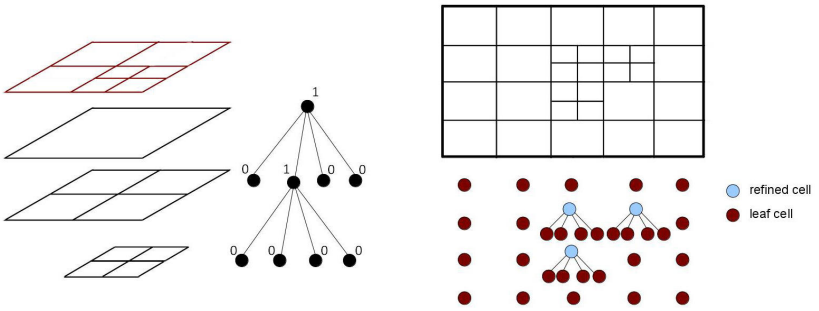

Fig. 2 Simple examples for tree-structured adaptive Cartesian grids represented either by a single cell-tree (left) or a forest of trees such as in p4est [13] and our parallel implementation of adaptive Cartesian grids in Sundance. Pictures taken from [7].

for an accurate discretization of the partial differential equation to be solved or for a good approximation of the domain boundary. Figure 3 shows an adaptively refined grid representing a fluid domain around a spherical obstacle. This example also shows the straight-forward adaptivity process that can follow geometry information, i.e., intersection of cells by the domain boundary, adaptivity criteria, or a priori user information on domains of particular interest. If the grid is implemented using sophisticated data algorithms, adaptivity can even be done in a way that does not destroy data locality [45].

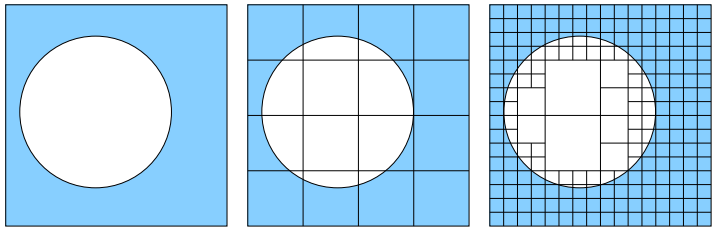

Fig. 3 Quadtree grid generation for a fluid domain around a spherical obstacle. Grid cells are further refined in this example if they are either cut by the geometry boundary or inside the fluid domain. This results in a regular refinement in the fluid domain. For visualization purposes, we show only every second refinement step. The whole grid generation corresponds to only one traversal of the final grid or its corresponding cell tree, respectively.

An additional advantage of adaptive Cartesian grids for application scenarios with moving geometries has already been mentioned in the introduction: a high flexibility for large geometry or even topology changes in an Eulerian, i.e., fixed grid setting. For these cases, we use immersed boundary methods to approximate the geometry, the imposed boundary conditions and the differential operators in grid cells intersected by the domain boundary with suitable accuracy. Our immersed boundary

approach is described in Sect. 6 in more detail. For the parallelization or domain partitioning of tree-structured adaptive Cartesian grids, space-filling curves have been shown to be a very cheap tool still providing quasi-minimal domain surfaces and, thus, communication costs [17, 12].

Mesh Generation and Storage in Sundance. A tree-structured Cartesian mesh can be stored in a highly efficient way as illustrated above. However, the Sundance mesh interface allows in principle for random access to mesh entities according to their IDs. A memory-saving tree-storage of the grid can not efficiently answer such arbitrary queries as each query would require a whole tree-traversal in the worst case. As a compromise, we linearize the tree and store all information of faces, edges, and vertices of all grid entities. As we can still use constant (up to scaling) element matrices and need only few interpolation matrices for elements with hanging nodes, we still save a considerable amount of memory if compared to unstructured grids. Table 1 compares the memory requirements of a Poisson solver with linear elements on tree-structured Cartesian and unstructured simplex meshes.

mesh	mesh storage	total memory
Cartesian mesh	783 MByte	1,600 MByte
Unstructured mesh	2,700 MByte	3,200 MByte

Table 1 Storage requirements of a three-dimensional Poisson solver using our implementation of a tree-structured Cartesian grid and an unstructured grid with a resolution of $531,441$ ($81 \times 81 \times 81$) quad elements (results from [7]).

Table 1 shows that fitting a structured mesh into an arbitrary access context of a PDE framework usually deteriorates the theoretical memory saving potential. In a tree-traversal oriented implementation such as in Peano [45], the memory requirements are $50-100$ times lower. However, grid generation is still done in a very simple and efficient way for our structured grids in a recursive refinement process, where the grid generator only needs to be given the information whether to locally refine further or not. This information can be derived from the geometry (refine only cells intersected by the domain boundary, e.g.), from error estimators, or from a priori given user input. The saving factor in terms of runtime strongly depends on the unstructured mesh generator and the scenario, but is obviously expected to be tremendously high for complex examples. Just remember that generating a Cartesian tree-structured grid corresponds to only a single grid traversal.

Introducing Rectangular Elements in Sundance. Sundance originally provided only simplex meshes. Thus, the first step towards adaptive Cartesian grids was the implementation of an additional, rectangular element including degrees of freedom located at the cell's vertices, edges, and faces. As finite element basis functions, we use the classical Lagrangian polynomial. The definition of a new basis only requires to implement the Sundance interface for basis function evaluation returning the basis function value for any given point on the reference element: In addition, the spatial derivatives are computed by automated differentiation.

Hanging Nodes and Pre-Fill Transformation. In contrast to unstructured meshes, adaptive Cartesian meshes contain lower-dimensional mesh components (nodes, edges, faces), that are not a node, edge, or face for all neighbouring elements. Figure 4 shows a two-dimensional example with hanging nodes and edges. To ensure continuity of the approximate solution, hanging nodes, edges, or faces may not pocess degrees of freedom and, accordingly, do not have associated entries in the vector of unknowns approximating the solution of our PDE. However, for efficiency reasons, we use the same element matrices (up to suitable scaling with the mesh width) in all cells. Thus, element matrices using a value at a hanging node, e.g., have to be modified as illustrated in Fig. 4 using an interpolation corresponding to the used finite element basis of the hanging node value from neighbouring non-hanging father nodes. The interpolation information is stored in a square matrix $M_{e,B}$ describing the interpolation of global (non-hanging) degrees of freedom to local (including hanging) values. In the example of Fig. 4, matrix $M_{e,B}$ for the upper left cell would be

$$M_{e,B} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{1}{3} & 0 & \frac{2}{3} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{8}$$

for bilinear basis functions and a lexicographic ordering of cell vertices: the first line indicates that the value of the local degree of freedom at the lower left vertex of the cell is the same as the value of the corresponding global degree of freedom (this node is not hanging). The same holds for the two upper vertices (line three and four of $M_{e,B}$). From line two of $M_{e,B}$, we see that we get the local value (at the hanging node in the lower right corner of the cell) as an averaged mean of the corresponding global value (lower right corner of the father cell) and the value at the upper right corner of the cell itself.

Independent from the actual grid, only a constant and rather small number of different interpolation matrices exists. Thus, we store them in a set of matrices and add only the index of the respective matrix $M_{e,B}$ (see Eq. (8)) to an element e in the mesh that contains hanging nodes, edges, or faces. Mathematically, the transformation of the element matrix A_e reads

$$A_e^{new} = M_{e,B_T}^T A_e M_{e,B_U}, \tag{9}$$

where B_T denotes the basis of the test space and B_U the basis of the ansatz space. This can be seen as an additional step of the matrix assembly process to be executed before accumulating element matrix contributions to the global system matrix. Therefore, we call this step the pre-fill transformation, which can be done very fast and efficiently due to the low number of cases and the purely local and small transformation matrices[1]. For the realization of the pre-fill transformation, the Sundance mesh interface sketched shortly in Sect. 4 has been extended by the following functionalities:

[1] Note that we restrict our grids to 1-irregular tree-structured grids, which ensures that, if a face, edge, or vertex is hanging, the corresponding face, edge, or vertex of the father cell is not hanging.

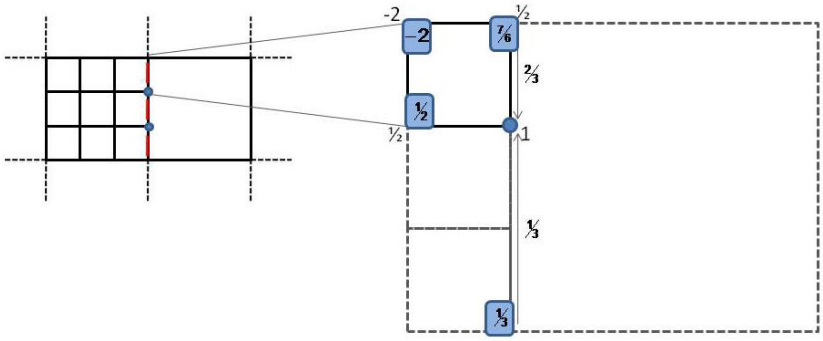

Fig. 4 Pre-fill transformation for a hanging node in a two-dimensional grid. Left: simple two-dimensional grid with hanging nodes (blue) and hanging faces (red); right: pre-fill transformation for element matrix lines associated to non-hanging element nodes interpolating the value at the hanging node from non-hanging father nodes. The resulting entries to be accumulated to the global system matrix are marked with blue rectangles.

1. return the information, if a face, edge, or vertex is hanging,
2. return the required face, edge, or vertex of the father cell for interpolation,
3. return the index of the cell within the parent cell.

The index of a cell within its father cell is required to determine the correct interpolation weights and indices of father cell entities.

Parallel Implementation of Cartesian Grids. The parallel implementation of a Cartesian grid has to provide two main ingredients: 1) a load balanced domain partitioning and 2) ghost layers for communication reduction. Once having these two ingredients, all other aspects such as the technical realization of data communication is taken care of by Sundance. For 1), we use the simple Z- or Morton-curve for grid partitioning. To further speed up the computations, we do the partitioning on a coarse grid level ignoring further refinement levels. This of course can lead to severe imbalances for grids with highly localized refinement and, in these cases, has to be substituted by a partitioning taking into account all grid levels. Figure 5 shows a very simple two-dimensional example for the partitioning of a grid into two subdomains.

Due to our pre-fill transformation algorithm, the second step, the determination of ghost cells, is a little more involved. We have to include also cells that do not have a face, edge, or vertex on the subdomain boundary for the reason illustrated in Fig. 5: grid elements at the subdomain boundary might have hanging facets and therefore contribute to the system matrix entries of neighbouring cells on a coarser level together with other child cells of the same father cell, which then might need information of a face of an additional coarser level neighbouring cell and so forth. However, the determination of ghost cells can still be done in a straightforward way by following such dependencies. In the current implementation, we still store

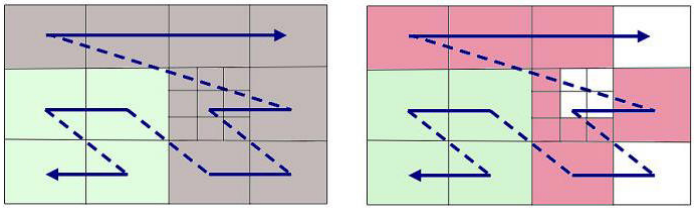

Fig. 5 Left: grid partitioning of a part of a very simple two-dimensional adaptive Cartesian grid into two subdomains using the Z-curve (Morton order) on a coarse grid level. Right: One of the two partitions (light green) with its required ghost cells (pink). Due to the pre-fill transformation, not only cells with a face on the subdomain boundary are required as ghost cells. Illustration taken from [7].

the whole grid information in every process, which obviously causes a too high memory overhead and has to be improved (currently work in progress). Table 2 shows strong scaling results for a porous media flow in a complex domain. It shows the scalability of the matrix assembly process for our Cartesian grid and the solver runtime, where matrix assembly scales better than the solver (TSF-BiCGStab solver from the Trilinos [20] library), which proves the adequacy of our implementation.

number of proc.	1	2	4	8	16	32	64	128
assembly (sec)	682	356	183	98	52	28	18	10
par. eff.	100%	96%	93%	87%	82%	76%	59%	53%
solver (sec)	875	497	238	136	91	49	36	21
par. eff.	100%	88%	92%	80%	60%	56%	38%	33%

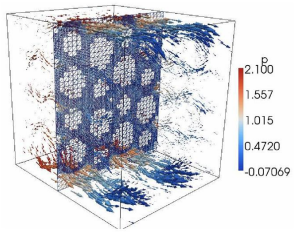

Table 2 Strong scaling results for a parallel three-dimensional solver of the Stokes-Brinkmann equation for porous media flow with varying porosity in a complex geometry with spherical low-porosity obstacles. The computations were performed on an adaptively refined Cartesian grid with $433,126$ cells and Q_1Q_1 elements (see right subfigure for a cut through the geometry and the grid) on the MPP cluster of the Leibniz Supercomputing Center (LRZ) in Garching (Opteron 2.6GHz AMD processors). As a system solver, we used the TSF-BiCGStab solver from the Trilinos [20] library.

After having the technical basis of Cartesian grids in the Sundance toolbox, we proceed with the description of the Nitsche method, which is the decisive step making the Cartesian grid implementation useful for accurate and efficient simulations.

6 Nitsche's Method for Fluid Dynamics in Moving Geometries

The general idea of Nitsche's method is the following: Let Γ_d be the Dirichlet boundary of $\Omega \in \mathbb{R}^n$ with boundary values g. This usually implies the state space $U_g = \{u \in U; u|_{\Gamma_d} = g\}$ and the space of test functions is $W_0 = \{w \in W; w|_d = 0\}$ with appropriate function spaces U and W. In this case, Dirichlet conditions are built into the function space U_g. The weak form is obtained by testing the strong equation with the test function and performing integration by parts where appropriate. The resulting boundary integrals usually vanish on Γ_d since $w|_{\Gamma_d} = 0$.

The Nitsche method works differently: The spaces U and W are used as state space and test function space and are oblivious of the boundary values g. Now, doing integration by parts, the boundary integral over Γ_d does no longer drop out. These integrals over Γ_d are symmetrized, i.e., if we have a summand

$$\int_{\Gamma_d} l\,(\partial_{\mathbf{n}} u, w)\,dS(x)$$

with a bilinear functional l, we replace it by

$$\int_{\Gamma_d} l\,(\partial_{\mathbf{n}} u, w)\,dS(x) + \int_{\Gamma_d} l\,(\partial_{\mathbf{n}} w, u)\,dS(x) - \int_{\Gamma_d} l\,(\partial_{\mathbf{n}} w, g)\,dS(x). \tag{10}$$

Note that for the exact solution u of our partial differential equation with Dirichlet boundary values g, the added terms cancel[2]. Furthermore, we add regularizations, also called penalty terms

$$\frac{\gamma}{h}\int_{\Gamma_d} uw dS(x) - \frac{\gamma}{h}\int_{\Gamma_d} gw dS(x) \tag{11}$$

and possibly

$$\frac{\gamma_2}{h}\int_{\Gamma_d} (u\cdot\mathbf{n})(w\cdot\mathbf{n}) dS(x) - \frac{\gamma_2}{h}\int_{\Gamma_d} (g\cdot\mathbf{n})(w\cdot\mathbf{n}) dS(x) \tag{12}$$

to coercify the problem and to enforce the Dirichlet conditions.

Nitsche's method for the Incompressible Navier-Stokes Equations For the formulation of Nitsche's method for the incompressible Navier-Stokes equations, we use the notation of Becker [4]. Let $\Omega \subset \mathbb{R}^2$ (or $\Omega \subset \mathbb{R}^3$) and $\mathbf{u} = (\mathbf{v}, p) \in H^1(\Omega)^2 \times L^2_0(\Omega)$ (or $H^1(\Omega)^3 \times L^2_0(\Omega)$) be the state space (velocity and pressure). Further, let $\Gamma = \partial\Omega, \mathbf{g} \in H^{1/2}(\Gamma)$, and $\mathbf{f} \in L^2(\Omega)^2$ (or $L^2(\Omega)^3$). We consider the stationary Navier-Stokes equations in a first step:

[2] The symmetrization is required to show that the error of the finite element solution minimizes a quadratic energy functional, which is used to prove the consistency order of the method (compare [34]).

$$-\nu\Delta\mathbf{v}+(\mathbf{v}\cdot\nabla)\mathbf{v}+\nabla p=\mathbf{f}\text{ in }\Omega, \tag{13}$$

$$\nabla\cdot\mathbf{v}=0\text{ in }\Omega, \tag{14}$$

$$\mathbf{v}=\mathbf{g}\text{ at }\Gamma. \tag{15}$$

In the following, we restrict to 2D for simplicity, although all methods are applicable also in 3D. For deriving a weak formulation, we test the PDE with $\Phi=(\Psi,\xi)\in H^1(\Omega)^2\times L^2_0(\Omega)$ and, as usual, integrate two terms by parts:

$$\begin{aligned}(-\nu\Delta\mathbf{v},\Psi)&=\nu\int_\Omega-[\nabla\cdot(\nabla\mathbf{v})]\cdot\Psi dx=\nu\int_\Omega\nabla\mathbf{v}:\nabla\Psi dx-\nu\int_\Gamma\partial_{\mathbf{n}}\mathbf{v}\cdot\Psi dS(x)\\&=\nu\,(\nabla\mathbf{v},\nabla\Psi)-\nu\,\langle\partial_{\mathbf{n}}\mathbf{v},\Psi\rangle\,,\end{aligned} \tag{16}$$

$$(\nabla p,\Phi)=-\int_\Omega p(\nabla\cdot\Psi)dx+\int_\Gamma p\mathbf{n}\cdot\Psi dS(x)=-\,(p,\nabla\cdot\Psi)+\langle p\mathbf{n},\Psi\rangle, \tag{17}$$

where $(\cdot,\cdot)$ denotes the scalar product introduced by the domain integral, $\langle\cdot,\cdot\rangle$ the scalar product introduced by the boundary integral. Summing up (16), (17), and the weak form of the convective term in (13), we thus get the usual distributed terms

$$a(\mathbf{u},\Phi)=\nu(\nabla\mathbf{v},\nabla\Psi)+((\mathbf{v}\cdot\nabla)\mathbf{v},\Psi)-(p,\nabla\cdot\Psi)-(\nabla\cdot\mathbf{v},\xi), \tag{18}$$

the right hand side term $(\mathbf{f},\Psi)$, and the new left hand side boundary terms

$$c(\mathbf{u},\Psi)=-\nu\langle\partial_{\mathbf{n}}\mathbf{v},\Psi\rangle+\langle p\mathbf{n},\Psi\rangle. \tag{19}$$

For symmetrization, we add $c(\Phi,\mathbf{v})$ on the left and compensate this by adding $c(\Phi,\mathbf{g})$ on the right (analogue to (10)). For stabilization and for enforcing the Dirichlet conditions (compare (11) and (12)), we add

$$\nu\frac{\gamma_1}{h}\langle\mathbf{v},\Psi\rangle+\frac{\gamma_2}{h}\langle\mathbf{v}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle$$

on the left and compensate this by adding

$$\nu\frac{\gamma_1}{h}\langle\mathbf{g},\Psi\rangle+\frac{\gamma_2}{h}\langle\mathbf{g}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle$$

on the right. Becker [4] uses a further stabilizing term $-\langle(\mathbf{vn})^-\mathbf{v},\Psi\rangle$, where $(\mathbf{vn})^-$ equals $\mathbf{vn}$ if $\mathbf{vn}<0$ and zero else. Taking all together, we have

$$\begin{aligned}&a(\mathbf{u},\Phi)+c(\mathbf{u},\Psi)+c(\Phi,\mathbf{v})+\nu\tfrac{\gamma_1}{h}\langle\mathbf{v},\Psi\rangle+\tfrac{\gamma_2}{h}\langle\mathbf{v}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle-\langle(\mathbf{vn})^-\mathbf{v},\Psi\rangle\\&=(\mathbf{f},\Psi)+c(\Phi,\mathbf{g})+\nu\tfrac{\gamma_1}{h}\langle\mathbf{g},\Psi\rangle+\tfrac{\gamma_2}{h}\langle\mathbf{g}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle-\langle(\mathbf{gn})^-\mathbf{g},\Psi\rangle\text{ for all }\Phi.\end{aligned} \tag{20}$$

If we consider the instationary Navier-Stokes equations

$$\frac{\partial \mathbf{v}}{\partial t} = \nu \Delta \mathbf{v} - (\mathbf{v}\cdot\nabla)\mathbf{v} - \nabla p + \mathbf{f} \text{ in } \Omega, \tag{21}$$

$$\nabla \cdot \mathbf{v} = 0 \text{ in } \Omega, \tag{22}$$

$$\mathbf{v} = \mathbf{g} \text{ at } \Gamma, \tag{23}$$

we have to add the time-derivative to the weak-form formulation. Doing this in the classical way, i.e., not using a space-time finite element approach, yields

$$\begin{aligned}\left(\tfrac{\partial \mathbf{v}}{\partial t}, \Psi\right) + a(\mathbf{u},\Phi) + c(\mathbf{u},\Psi) + c(\Phi,\mathbf{v}) + \nu\tfrac{\gamma_1}{h}\langle \mathbf{v},\Psi\rangle + \tfrac{\gamma_2}{h}\langle \mathbf{v}\cdot\mathbf{n}, \Psi\cdot\mathbf{n}\rangle - \langle (\mathbf{v}\mathbf{n})^{-}\mathbf{v},\Psi\rangle \\ = (\mathbf{f},\Psi) + c(\Phi,\mathbf{g}) + \nu\tfrac{\gamma_1}{h}\langle \mathbf{g},\Psi\rangle + \tfrac{\gamma_2}{h}\langle \mathbf{g}\cdot\mathbf{n}, \Psi\cdot\mathbf{n}\rangle - \langle (\mathbf{g}\mathbf{n})^{-}\mathbf{g},\Psi\rangle \text{ for all } \Phi. \end{aligned} \tag{24}$$

For time-discretization, we can use any finite difference scheme such as Euler or Runge-Kutta.

Issues with Moving Geometries Whereas the generalization of Nitsche's method to transient problems is straight-forward, the treatment of problems with moving or changing geometries is more involved in our fixed grid setting. To demonstrate the principle problem, we consider an explicit Euler time-stepping scheme applied to the Nitsche formulation (24):

$$\begin{aligned}\left(\mathbf{v}^{(k+1)},\Psi\right) = \left(\mathbf{v}^{(k)},\Psi\right) + dt\cdot\Big[&-\tilde{a}(\mathbf{u}^{(k)},\mathbf{u}^{(k+1)},\Phi) - c(\mathbf{u}^{(k)},\Psi) - c(\Phi,\mathbf{v}^{(k)}) \\ &-\nu\frac{\gamma_1}{h}\langle \mathbf{v}^{(k)},\Psi\rangle - \frac{\gamma_2}{h}\langle \mathbf{v}^{(k)}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle + \langle (\mathbf{v}^{(k)}\mathbf{n})^{-}\mathbf{v}^{(k)},\Psi\rangle \\ &+(\mathbf{f}^{(k)},\Psi) + c(\Phi,\mathbf{g}^{(k)}) + \nu\frac{\gamma_1}{h}\langle \mathbf{g}^{(k)},\Psi\rangle + \frac{\gamma_2}{h}\langle \mathbf{g}^{(k)}\cdot\mathbf{n},\Psi\cdot\mathbf{n}\rangle \\ &-\langle (\mathbf{g}^{(k)}\mathbf{n})^{-}\mathbf{g}^{(k)},\Psi\rangle\Big] \text{ for all } \Phi, \end{aligned} \tag{25}$$

where

$$\tilde{a}(\mathbf{u}^{(k)},\mathbf{u}^{(k+1)},\Phi) = \nu(\nabla\mathbf{v}^{(k)},\nabla\Psi) + ((\mathbf{v}^{(k)}\cdot\nabla)\mathbf{v}^{(k)},\Psi) - (p,\nabla\cdot\Psi) + (\nabla\mathbf{v}^{(k+1)},\xi).$$

Evaluating (25) for the basis functions $\Phi_i = (\Psi_i, 0)$ and $\Phi_{N+i} = (0,\xi_i), i = 1,\ldots,N$ leads to the system of equations for time step $t^{(k)} \to t^{(k+1)}$. We get the explicit Euler with Chorin's projection method for the pressure by replacing $\mathbf{v}^{(k+1)}$ in the last N equations (discrete continuity equation) by the formula given in the first N equations (discrete momentum equations). As we compute a solution at time step $t^{(k+1)}$, we have to integrate expressions involving $\mathbf{v}^{(k)}$ over $\Omega^{(k+1)}$. However, $\mathbf{v}^{(k)}$ does not provide physically meaningful values in $\Omega^{(k+1)} \setminus \Omega^{(k)}$. To achieve a consistent formulation, one possibility would be to apply space-time finite element formulations [43, 5]. This, however, would require four-dimensional cut-cell quadrature functionality for spatially three-dimensional problems. Thus, we solve the moving geometry problem with a simpler approach in a first step: We restrict ourselves to fully implicit time-stepping schemes and, thus, minimize the impact of values of $\mathbf{v}^{(k)}$ in $\Omega^{(k+1)} \setminus \Omega^{(k)}$. In addition, we prevent $\mathbf{v}^{(k)}$ from taking extreme values outside $\Omega^{(k)}$ by solving a Poisson equation with a small weighting factor in the fictitious domain

(compare [7]). The results presented in Sect. 8 and 9 show the usability of this simple approach for a suite of fluid-structure interaction benchmarks from [22, 23] and for shape optimization.

7 Implementation in Sundance

To implement Nitsche's method for flow equations on complex and moving domains, we first need some technical ingredients, i.e., volume integrals over cut cells integrating only over the part of a grid element that is inside the fluid domain and boundary integrals over the domain boundary. Both require an explicit representation of the domain boundary. As we work with second-order discretizations, we can use simple polygons (in 2D)[3] or surface triangulations (in 3D). Figure 6 illustrates the representation of domains and cut cell volumes. In 2D, we allow a cell to be intersected by an arbitrary polygon. In this case, we can always find a suitable decomposition of the remaining fluid part of a cell into simple trapezoidals and triangles. In 3D, the analogue process with arbitrary surface triangulations intersecting a cubic grid element would lead to a too large amount of different cases and, thus, is not feasible. Therefore, we restrict to cases, where all cell edges are intersected at most once by the surface triangulation. Connecting these intersection points by straight lines leads to a new, approximate, but still second-order accurate surface triangulation and only a small number of cases as displayed in Fig. 6. In both cases, 2D and 3D, we end up with a decomposition of the fluid part of the cut cell into simple elements $E_i, i \in I_E$ of given type that can be integrated up to machine precision with suitable quadrature rules. The same holds for the surface integrals.

For the volume integrals over cut cells, we thus have

$$\int_{E\cap\Omega} f(x)dx = \sum_{i\in I_E} \int_{E_i} f(x)dx = \sum_{i\in I_E} \sum_{j=1}^{N_i} \omega_{i,j} f(x_{i,j})dx, \tag{26}$$

where $E_i, i \in I_E$, are the simple subcells into which we decompose of the fluid part of the cell and $x_{i,j}, j = 1,\ldots,N_i$, are the quadrature points used in subcell i. With this formula, we can compute all cut cell integrals up to machine precision. However, this method can lead to a large number of quadrature points $x_{i,j}$ in particular in 3D and for higher order basis functions. Therefore, we reduce the number of quadrature points by using precomputed weights

$$\omega_k = \sum_{i\in I_E} \sum_{j=1}^{N_i} \omega_{i,j} l_k(x_{i,j}), k = 1,\ldots,K \tag{27}$$

[3] For 2D geometries, there is also an implementation in Sundance that can deal with analytical geometry representations (such as a circle) that are then automatically approximated by a suitable polygon internally [7].

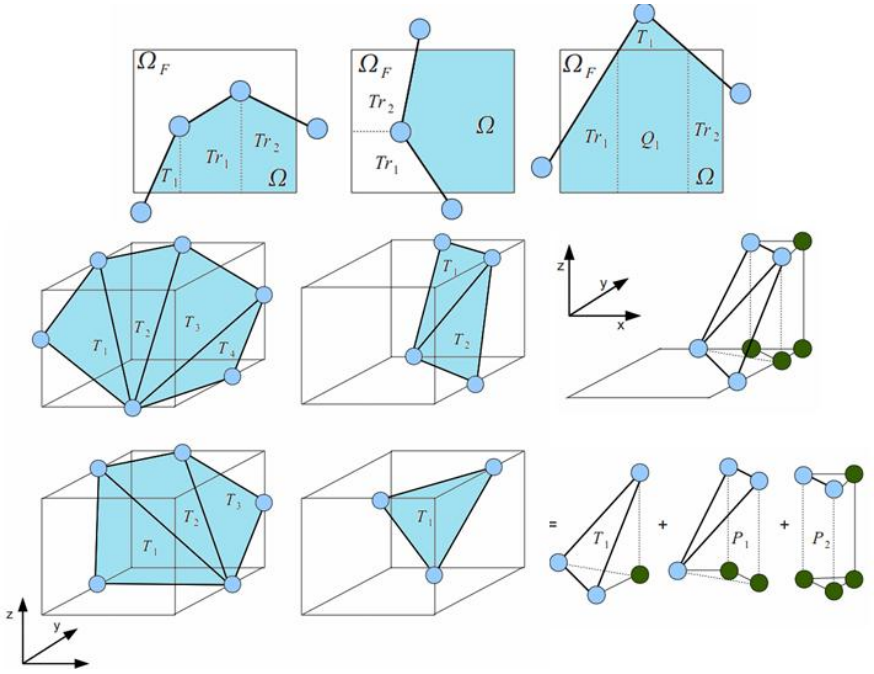

Fig. 6 Illustration of the decomposition of the geometry boundary and the fluid parts of a cut cell in two-dimensional and three-dimensional cases. This method allows for the computation of the finite element integrals in cut cells and on boundaries described by polygons in 2D and triangulations in 3D with only a small set of quadrature rules for simple geometric elements. Illustrations taken from [7].

where l_k denotes the Lagrangian polynomial associated to the point x_k, i.e., $l_k(x_k) = 1, l_k(x_m) = 0$ for all $m \in \{1,\ldots,K\} \setminus \{k\}$. The quadrature rule in (27) and, thus, $N_i, i \in I_E$, have to be chosen such that all Langrange polynomials l_k are integrated exactly. Since our function f is a product of basis functions and their derivatives, we have for a suitable K by

$$f(x) = \sum_{k=1}^{K} f(x_k) l_k(x).$$

Thus, we can replace (26) by a quadrature rule with the precomputed weights ω_k:

$$\int_{E\cap\Omega} f(x)dx = \sum_{i\in I_E} \sum_{j=1}^{N_i} \omega_{i,j} f(x_k) l_k(x_{i,j})dx = \sum_{k=1}^{K} \omega_k f(x_k). \tag{28}$$

Here, the number of new quadrature points K only depends on the approximation order of the finite element basis and the considered PDE but *not* on the function $f(x)$. This fits well with the technical restriction given by Sundance that a constant

number of quadrature points is to be used throughout the whole computational grid. Whenever the domain Ω changes, the weights ω_k have to be recomputed in a loop over all cut cells computing the local quadrature points $x_{i,j}$ for each subcell on the fly.

Line or surface boundary integrals are computed analogously. However, we do not use precomputed weights here as 1) some boundary integrals involve normal vectors that are different for each subpart of the boundary inside a given cell, 2) boundary integrals are cheaper such that we can afford storing all quadrature points, 3) three-dimensional cases require the usage of data points in the vicinity but not directly on the boundary due to the approximation of the original triangulation within each cut cell.

Technically, the implementation of Nitsche's method in Sundance requires new user interfaces for

- description of the geometry by a polygon/triangulation
- cell filters selecting intersected cells,
- integrals over cut cells, and
- boundary integrals.

To realize moving boundaries in Sundance, several data have to be updated after changing of the geometry:

- the positions of the surface polygon or triangulation,
- internal information such as IDs of cells cut by components of the surface object,
- precomputed quadrature weights for the finite element integration during matrix assembly,
- entries of the system matrix/matrices.

To move surface nodes, we use the local surface velocity (given by the boundary conditions) at the respective position. The update of all internal geometry information is hidden from the user in the function `.update()`. The function `.reAssembleProblem()` recomputes quadrature weights and assembles the new system matrix/matrices.

8 Fluid-Structure Simulations Using Nitsche's Method

The suitability and accuracy of our method using Cartesian grids in combination with Nitsche's method for simple model problems and pure flow simulation in static and moving domains has been shown in [7, 8]. In this paper, we focus on results obtained for a multiphysics application class, i.e., fluid-structure interactions.

Our Partitioned Approach for Coupled Fluid-Structure Simulation in Sundance. We use the incompressible Navier-Stokes equations (1) and (2) and combine them with the structural equation (3) and the coupling conditions (4) and (5) to a full fluid-structure simulation environment.

As we use an Eulerian fixed grid approach as described in the previous sections for the fluid part, but the common Lagrangian approach with a classical enforcement of boundary conditions for the structure part, we have to couple the interface displacements in addition:

$$\Gamma_F = \{x + \mathbf{u}_s | x \in \Gamma_L\}, \tag{29}$$

where Γ_F is the internal moving geometry description (polygon or surface triangulation) of the fluid solver and Γ_L the Lagrangian structure surface description (constant throughout the simulation). See Fig. 7 for an illustration. Technically, this is realized in Sundance using twin polygons or twin triangulations. The nodes of these two polygons or triangulations are connected via a bijective mapping from the node set of the first polygon/triangulation to the node set of the second polygon/triangulation as illustrated in Fig. 7 by the dashed lines. Whenever a value of an unknown function is changed on a node of one polygon/triangulation, it is automatically also changed on the respective node of the twin polygon/triangulation.

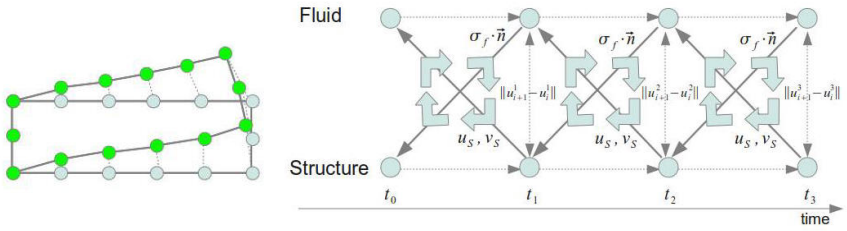

Fig. 7 Left: Eulerian (fluid) and Lagrangian (structure) view of a simple two-dimensional moving structure. The green circles mark the nodes of the polygonal geometry description in the fluid solver, whereas the light grey circles are the boundary vertices of the Lagrangian reference domain on the structure side. Right: Schematic view of the staggered coupling approach for partitioned fluid-structure interaction simulations. After executing a time step of the flow solver, forces at (29) are transfered to the structure solver, which subsequently executes its time step and produces displacements and velocities at the wet surface as an output, which serve as a boundary condition at (29) for the next fluid step and so forth. This process is repeated until the residual of the fix-point equation for the displacements falls below a given threshold. Illustrations taken from [7].

As a time discretization, we use implicit Euler for the fluid and structure equations. To perform a time step of the whole coupled problem, the partitioned approach requires an iterative execution of fluid and structure solver steps involving an exchange of boundary values. We use the widespread staggered coupling approach as shown in Fig. 7. For incompressible fluids, this coupling in general leads to unstable time stepping if used in an explicit manner (only one fluid and structure step per time step). Thus, we have to iteratively repeat the fluid and structure steps until the residual of the associated fixed point equation for the displacements at the wet surface falls below a certain threshold. To achieve convergence of the fluid-structure iterations, we use Aitken underrelaxation when transferring displacements from the structure solver to the flow solver. Aitken underrelaxation determines an

adaptive scalar underrelaxation factor in each iteration [24]. More sophisticated coupling methods such as interface quasi-Newton schemes [14] or multilevel [46, 9] approaches yield an even faster convergence. However, for our purpose of showing the potential of Nitsche's method for the simulation of fluid-structure interaction on structured Cartesian grids, Aitken underrelaxation suffices.

In the following, we present some of our numerical results for fluid-structure interaction scenarios in 2D and 3D. Further results can be found in [7].

Rigid Body Motion in 3D. Our first scenario is the three-dimensional rigid body motion of a spherical object in a flow channel as shown in Fig. 8. In a channel of dimensionless size $8 \times 4 \times 4$, a sphere with dimensionless radius one and a dimensionless mass of one moves according to Newton's law of motion and the sum of forces exerted on it by the fluid flow. The fluid has the dimensionless density one and the inflow speed is set to 2.25. We keep the sphere fixed until dimensionless time $T = 0.2$. After that initial period, the sphere is released and accelerated by the surrounding fluid flow. The fluid flow is simulated on a very coarse regular Cartesian mesh with only $13 \times 9 \times 9$ elements, whereas the sphere's surface is approximated by 500 triangles with 252 nodes. Due to the relatively large structure density and the rather small radius of the sphere, this example allowed for explicit coupling, i.e., we performed only one fluid and structure step in each time step. Figure 8 shows the good solution quality in spite of the very coarse fluid mesh resolution. The sphere exhibits a smooth movement in flow direction as expected for a correct solution. The forces in x-direction show some unphysical peaks due to the time integration issues for moving geometries mentioned above. Depending on the the state of the simulation, these peaks can even result in negative forces slowing down the motion of the sphere in x-direction. However, each of these peaks decays very fast and, thus, does not influence the overall solution significantly. The height of the peaks obviously depends on the time step size and the spatial resolution.

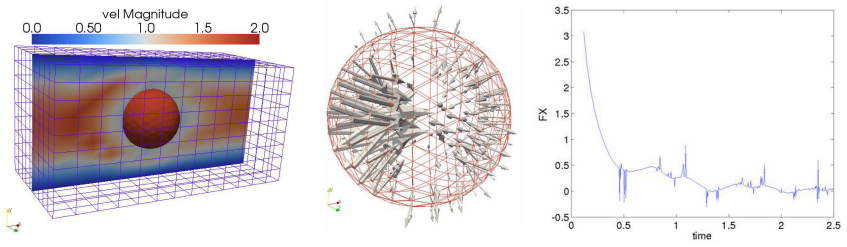

Fig. 8 Simulation of a rigid body motion of a sphere in a fluid flow using a very coarse Cartesian mesh. Left: absolute velocities in a plane of the fluid domain; middle: force vectors acting on the sphere; right: total forces in flow direction acting on the sphere plotted over time. The forces decay to zero as the sphere accelerates. Illustrations taken from [7].

Two-Dimensional Benchmarks. To verify our implementation by more reliable data than heuristic observations as in the 3D example above, we simulated

2D benchmark scenarios for fluid-structure interaction proposed in [22, 23]. The scenario consists of a channel flow with an embedded spherical obstacle and an attached flexible bar structure. The spherical obstacle itself remains fixed, but the bar bends in a steady state (FSI1) or oscillating (FSI3) manner depending on the Reynolds number of the channel flow. Due to a slight offset in vertical direction, the structure is exposed to both horizontal and vertical forces. Figure 9 shows the geometric setup of the benchmark. The structure is a super-elastic compressible material with $\rho_s = 10^3 \frac{kg}{m^3}$ and Young's modulus $E = 1.4 \cdot 10^6 \frac{kg}{m^3}$. We discretize the Navier-Stokes equations using Q_2Q_1 and the structure using Q_1 elements. For this scenario, an implicit coupling with several fluid-structure iterations is mandatory to achieve stable results. We use $\varepsilon = 10^{-6}$ as a stopping criterion for our staggered fluid-structure iterations with a constant underrelaxation factor $\omega = 0.3$.

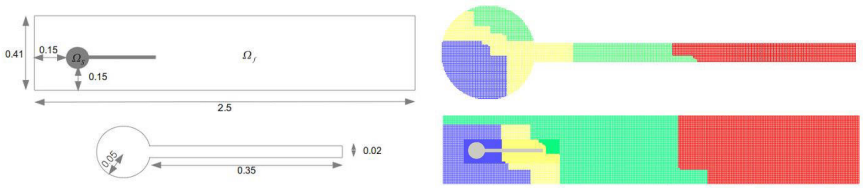

Fig. 9 Two-dimensional fluid-structure interaction benchmark scenario from [22, 23]. The scenario shows a fixed and rigid spherical object with an attached super-elastic compressible flap structure embedded in a channel flow. Left: Scenario geometry with a slight offset of the structure from the channel axis. Right: domain decomposition of the structure and the fluid part for a parallel simulation with four processors. Illustrations taken from [7].

The FSI1 scenario is a stationary case with a steady state flow and a slight constant bending of the elastic bar structure in y-direction. The inflow velocity is set to $v_f = 0.2 \frac{m}{s}$. Convergence within a time step was achieved after 20 iterations on average. Table 3 shows the computed values for the displacements of the tip of the bar in x- and y-direction as well as the total drag and lift forces exerted on the structure by the fluid. The total forces have been calculated by curve integrals on the polygon. The table shows already a good agreement with the benchmark values for the coarsest mesh resolution and a convergence towards the correct values with increasing resolution. The simulations for this scenario have been performed in parallel using four processors for each, fluid and structure solver, and two processors for the surface polygon. The associated domain decomposition of the fluid and structure domain is shown in Fig. 9. As a solver, we used the SuperLU-DIST linear solver within an outer non-linear NOX solver[4]. This combination resulted in an acceptable but not yet optimal parallel efficiency of 54% [7]. Note that the simple load balancing strategy, a quasi-serial phase for the surface polygon, and further optimization potential in the storage of grid partitiones currently limits the scalability, a fact that shall be improved in further development steps.

[4] trilinos.sandia.gov/packages/nox/

cells$_{fluid}$	**cells**$_{struct}$	**lines**$_{poly}$	**displ. x**	**displ. y**	**drag**	**lift**
2,610	3,380	136	$1.86 \cdot 10^{-5}$(18%)	$1.230 \cdot 10^{-3}$(50%)	14.078(1.5%)	0.8202(7.4%)
36,666	13,370	1,093	$2.21 \cdot 10^{-5}$(2.6%)	$0.703 \cdot 10^{-3}$(14%)	14.198(0.7%)	0.8101(6.1%)
54,104	13,370	1,093	$2.18 \cdot 10^{-5}$(3.5%)	$0.846 \cdot 10^{-3}$(5.2%)	14.224(0.5%)	0.7930(3.8%)
ref. values			$2.27 \cdot 10^{-5}$	$0.821 \cdot 10^{-3}$	14.295	0.7638

Table 3 Simulation results for two different mesh resolutions and the benchmark scenario FSI3 from [22, 23]. The displacement of the tip of the bar in x- and y-direction, the total drag, and the total lift force are compared with the benchmark reference values in [23]. Numbers in parentheses denote the absolute value of the relative error.

The FSI3 is a transient case with a higher inflow velocity of $2\frac{m}{s}$ that leads to an oscillatory movement of the flexible bar and also requires a strong, i.e., implicit coupling of fluid and structure. We used Aitken underrelaxation resulting in 11 iterations per time step on average. Figure 10 and Tab. 4 show the resulting movement of the bar and the respective tip displacements, total drag, and total lift with their mean value, the amplitude of their oscillation, and their frequency. The comparison with the benchmark reference values from [23] shows a good agreement. However, for more reliable convergence statements, additional mesh resolutions have to be examined in addition in future work.

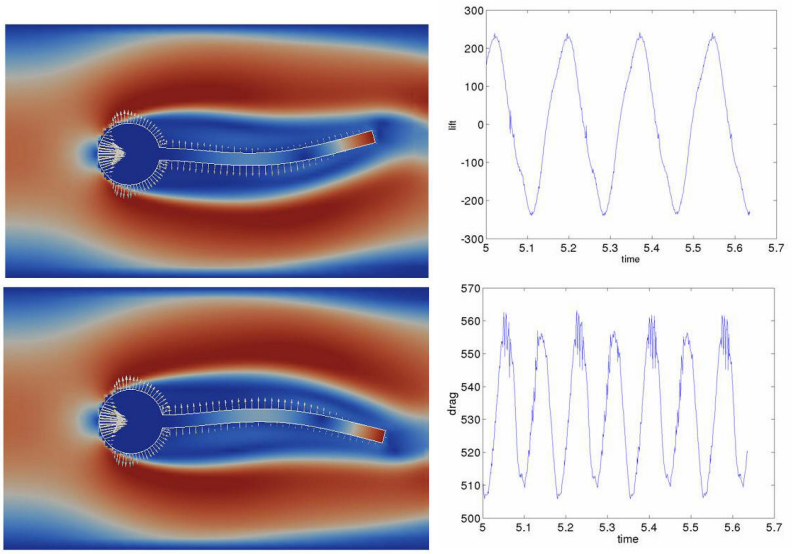

Fig. 10 Simulation results for the transient FSI3 benchmark from [22, 23]. Left: two snapshots of the solution with colours representing the absolute fluid velocities and the structural displacement. Arrows show the forces acting on the structure; middle: time variant total drag; right: time variant total lift. Plots taken from [7].

$\textbf{cells}_{fluid}$	$\textbf{cells}_{struct}$	$\textbf{lines}_{poly}$	**displ. x**	**displ. y**	**drag**	**lift**
2,507	864	136	-0.02866 $\pm 0.02857[11.1]$	-0.00111 $\pm 0.03301[5.5]$	524.5 $\pm 23.5[11.1]$	56.50 $\pm 214.50[5.5]$
5,133	864	136	-0.00288 $\pm 0.00282[11.4]$	0.00166 $\pm 0.03452[5.7]$	532 $\pm 25[11.4]$	-1.5 $\pm 229.50[5.7]$
ref. values			-0.00269 $\pm 0.00253[10.9]$	0.00148 $\pm 0.03438[5.3]$	457.3 $\pm 22.66[10.9]$	2.2 $\pm 149.78[5.3]$

Table 4 Simulation results for two different mesh resolutions and the benchmark scenario FSI3 from [22, 23]. For all measured values, i.e., displacement of the tip of the bar in x- and y-direction, total drag, and total lift force, offset, amplitude, and frequency (in $[\cdot]$) are measured and compared with the benchmark reference values in [23].

Three-Dimensional Bending Tower. As a three-dimensional example for the coupled simulation of fluid-structure interactions with a moving flexible structure, we consider a channel flow with a bending tower (see Fig. 11). The fluid has a again density of $10^3 \frac{kg}{m^3}$ and a parabolic inflow profile with maximal velocity $0.45 \frac{m}{s}$. The bending tower is modeled by a super-elastic material with Young's modulus $E = 0.4 \cdot 10^6 \frac{kg}{ms^2}$. The tower has a size of $0.05m \times 0.25m \times 0.1m$. We use an adaptive Cartesian grid for the flow solver with adaptive refinement in a rectangular domain around the tower. The resolution of the coarse background mesh is $27 \times 7 \times 7$ cells. We discretized the Navier-Stokes equations with stabilized Q_1Q_1 and the structure equation with Q_1 elements. The stabilized Q_1Q_1 are known to work well in three-dimensional fluid benchmarks, e.g., in [40]. The coupling iterations are performed with an underrelaxation factor $\omega = 0.3$ up to an error tolerance $\varepsilon = 10^{-6}$. Figure 11 shows the resulting bending of the tower, which can not be verified by benchmark values but gives reasonable results [7].

9 Shape Optimization with the Immersed Boundary Approach

We now demonstrate the application of our immersed boundary approach to shape optimization problems. To this end, we consider the Stokes equations

$$-\nu \Delta \mathbf{v} + \nabla p = 0, \quad \operatorname{div} \mathbf{v} = 0, \quad \mathbf{v}|_{\partial\Omega \setminus \Gamma} = d, \quad \mathbf{v}|_{\Gamma} = 0. \tag{30}$$

Here, $\Omega = \Omega_C \setminus B$, where Ω_C is a rectangular channel and B is a body with boundary Γ. The goal is to find a design for B such that the cost functional

$$J(\mathbf{v}, \Omega) = \nu \int_{\Omega} \|\nabla \mathbf{v}\|^2 \, dx, \tag{31}$$

which is related to the drag of the body, is minimized. Here, B shall have a prescribed volume and (except possibly at the front and rear tip points) a sufficiently smooth boundary. The latter condition is required to ensure existence of optimal shapes.

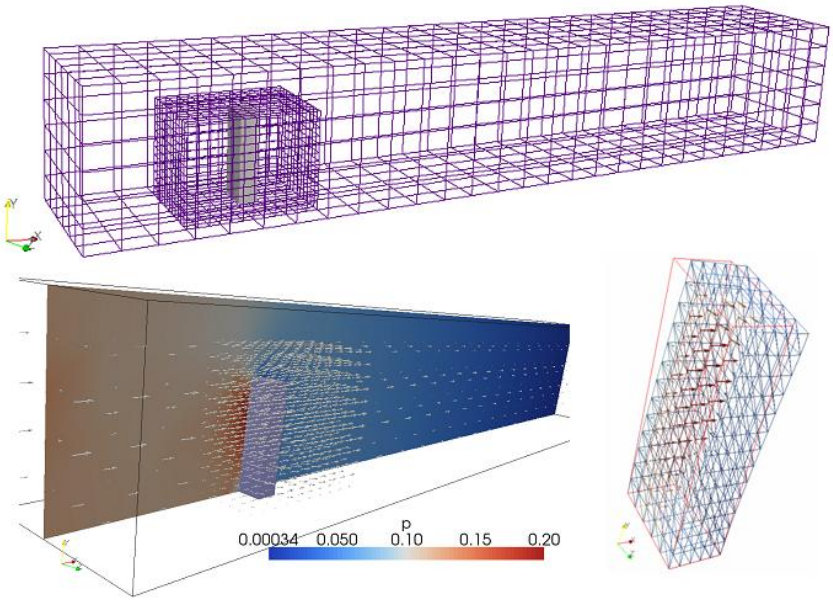

Fig. 11 Three-dimensional simulation of a bending tower in a channel flow. The upper picture shows the tower and the surrounding computational grid with an adaptive refinement in a block around the tower. Left: flow field and fluid pressure; right: bending of the tower. Illustrations taken from [7].

Denoting by $(\mathbf{v}(\Omega), p(\Omega))$ the solution of the Stokes equations on the domain Ω, the shape derivative of $j(\Omega) := J(\mathbf{v}(\Omega), \Omega)$ can be shown to satisfy

$$dj(\Omega)[V] = -\nu \int_\Gamma \|\partial_n \mathbf{v}(\Omega)\|^2 V \cdot n \, dS(x), \tag{32}$$

see [32, pp. 29/30]. A similar result holds for the more general case of the Navier-Stokes equations, but it then involves the adjoint state [32, pp. 31/32]. The Stokes equations in combination with the cost function (31) allow to express all occurrences of the adjoint state in terms of $\mathbf{v}$, thus making the adjoint state superfluous.

The shape of the body B is described by a closed polygon with nodes (x_0, y_0), …, (x_{2m-1}, y_{2m-1}). Here, the next point (x_{2m}, y_{2m}) would correspond to (x_0, y_0). We only consider shapes that are symmetric with respect to a horizontal center line (parallel to the x-axis):

$$y_0 = y_m, \quad x_i = x_{2m-i}, \;\; y_0 - y_i = y_{2m-i} - y_0 \;\; (1 \le i < m).$$

We do not work directly with this representation, but use polar coordinates with respect to $(a, b) = ((x_0 + x_m)/2, y_0)$. For $i = 0, \ldots, m$, the i-th point is allowed to move

along the radial ray $(a,b)+\mathbb{R}(x_i-a,y_i-b)$. The transformed point is parameterized by the radial scaling r_i: $(x_i(r_i),y_i(r_i)) = (a,b)+r_i(x_i-a,y_i-b)$. Due to our symmetry assumptions, specifying $r=(r_0,\ldots,r_m)^T$ uniquely characterizes the shape of the polygonal boundary of the deformed body.

In the function space setting, the existence of optimal shapes requires coercivity of the cost function in a sufficiently strong shape space. In the discrete setting, we enforce some amount of smoothness by adding an H^1-like regularization for r to the cost function (31). Also, we work with an H^1-like inner product by switching to the parameterization $z\in\mathbb{R}^{m+1}$ with $r_i=\sum_{j=0}^{i} z_j$. We then have $z_0=r_0$ and $z_i=r_i-r_{i-1}$, $i>0$. In our computations, we work with the body $B(z)$ parameterized by z. We then can use an l_2-norm for the regularization:

$$\gamma m\sum_{i=1}^{m} z_i^2. \tag{33}$$

The factor m stems from the fact that if we view the r_i as evaluations of a function $r(\tau)$ on an equidistant grid $\tau_0,\ldots,\tau_m$ with $\tau_i=i\delta$, $\delta=1/m$, then $mz_i=(r_i-r_{i-1})/\delta$ approximates $r'(\tau_i-\delta/2)$. Hence,

$$\int_0^1 r'(\tau)^2\,d\tau \approx \delta\sum_{i=1}^{m} r'(\tau_i-\delta/2)^2 \approx m\sum_{i=1}^{m} z_i^2. \tag{34}$$

For evaluating the discrete version $j_h(z)$ of the cost function (31) and its gradient $\nabla j_h(z)$ at a point z in parameter space, we solve the Stokes equation, discretized by Nitsche's method, on $\Omega(z)=\Omega_C\setminus B(z)$ to obtain the discrete velocity field $\mathbf{v}_h$. Then

$$J(\mathbf{v}_h,\Omega(z)) = \nu\int_{\Omega(z)} \|\nabla\mathbf{v}_h\|^2\,dx \tag{35}$$

is computed and the regularization (33) is added. For evaluating the gradient $\nabla j_h(z)$ at a point z we use that the map $z\mapsto(x_i(z),y_i(z))-(a,b)$ is linear. Thus, the direction of variation $e_i=(\delta_{ik})_{0\le k\le m}$ of z is linearly mapped to the direction of variation $(x_p(e_i),y_p(e_i))-(a,b)$ of the p-th polygon node $(x_p(z),y_p(z))$.

Therefore, for the z-variation direction e_i, the value of the corresponding displacement field direction $V=V^p$ at the polygon's pth node is $(x_p(e_i),y_p(e_i))-(a,b)$. To compute $\nabla_{z_i} j_h(z)$ we thus evaluate

$$-\nu\int_{\Gamma(z)} \|\partial_n\mathbf{v}_h\|^2\,V^p\cdot n\,dS(x). \tag{36}$$

The required methods, in particular the conversion of nodal values along the polygon into expressions that can appear in line integrands over the polygon curve have all been implemented by us for general use in Sundance.

We stress that the formula (36) for $\nabla_{z_i} j_h(z)$ is not fully exact since (32) requires the exact solution $\mathbf{v}$ of the PDE, whereas $\mathbf{v}_h$ is only a numerical approximation that,

in addition, depends not only on the shape of $\Omega(z)$ but also on the underlying FEM mesh.

In the numerical test presented here we require also symmetry with respect to the vertical center line passing through (a,b). To this end, we choose $m = 2l$ and only prescribe the lower left quarter of the profile, which is described by $z_0, \ldots, z_l$, where we require $x_l = a$. Thus, (x_0, y_0) lies on the horizontal center line and (x_l, y_l) lies on the vertical center line. The rest of the profile is obtained by symmetry and the center of mass is then automatically fixed to (a,b).

We impose a constant volume constraint $\mathrm{vol}(\Omega(z)) = c_{\mathrm{vol}}$, with c_{vol} denoting the volume of the initial shape. The volume and its derivative can be obtained conveniently by, e.g., the Leibniz sector formula. We also pose inequality constraints on the maximum radial elongation / contraction: $1/2 \leq r_i(z) \leq 2$ for all i. The bounds are chosen such that the constraints do not become active, but they can help avoiding too strong initial design changes during optimization.

We choose $\Omega_C = (0, 2.5) \times (0,1)$, $(a,b) = (0.75, 0.5)$, the tip point of the initial shape is at $(0.67, 0.5)$ and the initial volume is 0.0163672. We impose Dirichlet conditions $v = (0.2, 0)$ on the boundary of Ω_C and $v = 0$ on $\partial B(z)$. The boundary conditions on $\partial B(z)$ are enforced by Nitsche's method. Figure 12 shows the initial shape on the left. The boundary of the body is described by $2m = 4l = 80$ points, i.e., we have $m = 40$, $l = 20$. We use a Taylor-Hood discretization (biquadratic for the velocity, bilinear for the pressure) on a Cartesian mesh with hanging nodes. The grid consists of a 145×101 mesh, where the 23×41 subgrid centered around $(a,b) = (0.75, 0.5)$ is refined by cellwise trisection in both space dimensions to a obtain a 69×123 grid on this subregion. The regularization parameter is $\gamma = 0.006$.

We implemented a general purpose C++ interface that couples our Sundance simulation, which solves the Stokes equation and evaluates j_h, ∇j_h, etc., to the interior point optimization software IPOPT [44]. We use the limited memory BFGS approximation of the Lagrange function's Hessian matrix that is built into IPOPT.

As mentioned, due to the fact that the shape derivative formula is only exact in the continuous setting, the computed derivatives are not exact derivatives of the discretized cost function. Thus, we can achieve only a moderate (but sufficient) accuracy in satisfying the stopping criteria of the NLP solver IPOPT.

Starting with the initial shape shown on the left in Fig. 12, we obtain a solution (with the achievable accuracy due to inexact discrete shape gradient) after 10 IPOPT iterations. The objective function is reduced by 6.9 %. The optimal shape is shown on the right in Fig. 12. Note that the result depends on the chosen regularization.

10 Conclusion

We proposed an implementation of adaptive Cartesian grids in combination with Nitsche's method for an accurate enforcement of boundary conditions on complex and moving domains. Adaptive Cartesian meshes have severe advantages for an efficient implementation of solvers for partial differential equations and PDE con-

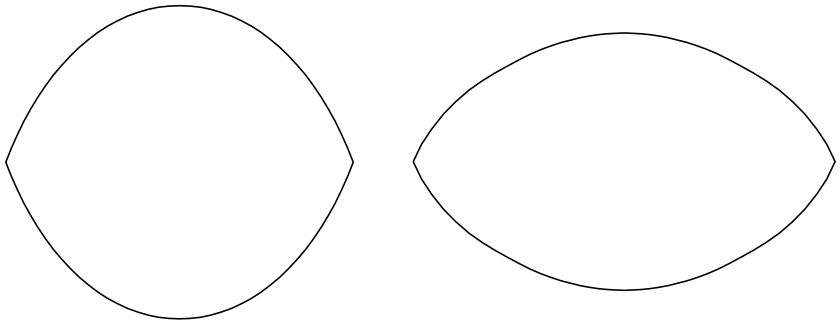

Fig. 12 Two-dimensional shape-optimization example for a flow around a rigid body. Left: initial shape; right: optimal shape.

straint optimization problems. These advantages reach from minimal memory requirements over efficient data access algorithms, easy load balanced domain decomposition using space-filling curves to a very efficient mesh generation and adaptive refinement process. In this paper, we implemented Cartesian grids in the Sundance toolbox designed for the fast, weak-form formulation based realization of PDE solvers and PDE constraint optimization algorithms. Sundance minimizes the user-effort for setting up a new application. In addition, it offers numerous solvers, e.g., by an interface to the Trilinos solver library. Thus, it is an ideal platform for the development and testing of finite element based methods for a higher-order representation of boundary conditions for Cartesian grids that inherently are not able to represent arbitrary complex and moving geometries with more than first order accuracy without further efforts. We used an old method known introduced by Nitsche in the 1970s that allows for a consistent weak enforcement of Dirichlet boundary conditions on boundaries cutting the Cartesian grid cells in an arbitrary way. We enhanced this method to flow simulations in moving geometries based on the incompressible Navier-Stokes equation and showed their accuracy even for cases of fluid-structure interactions, where the accuracy at the boundary between fluid and structure is particularly crucial, and for a shape-optimization example.

Fitting Cartesian grids to the software requirements of Sundance destroys some of their advantages. However, our implementation allows for a fast development and enhancement of methods such as the Nitsche method and, thus, lays the basis for a high-performance computing implementation. Future work is the investigation of methods limiting the condition number of system matrices resulting from Nitsche's method for very small cut cell fluid parts, the development of improved methods for moving geometries avoiding peaks in pressure or forces due to the extrapolation of previous time step solutions to the current time step's domain, and the examination of additional time-step restrictions induced by Nitsche's method.

References

1. I. Babuška. Numerical Solution of Boundary Value Problems by the Perturbed Variational Princile. *Technical Note BN-624*, 1969.
2. I. Babuška. The finite element method with penalty. *Mathematics of Computation*, 27:122–128, 1973.
3. Y. Bazilevs, C. Michler, V. M. Calo, and T. J. R. Hughes. Isogeometric variational multiscale modeling of wall-bounded turbulent flows with weakly enforced boundary conditions on unstretched meshes. *Computer Methods in Applied Mechanics and Engineering*, 199:780–790, 2010.
4. R. Becker. Mesh Adaption for Dirichlet Flow Control via Nitsche's Method. *Communication in Numerical Methods in Engineering*, 18:669–680, 2002.
5. M. Behr. Simplex space-time meshes in finite element simulations. *Int. J. Numer. Meth. Fluids*, 57:1421–1434, 2008.
6. J. A. Bello, E. Fernández-Cara, J. Lemoine, and J. Simon. The differentiability of the drag with respect to the variations of a Lipschitz domain in a Navier-Stokes flow. *SIAM J. Control Optim.*, 35(2):626–640, 1997.
7. J. Benk. *Immersed Boundary Methods within a PDE Toolbox on Distributed Memory Systems*. PhD thesis, Technische Universität München, 2012.
8. J. Benk, M. Mehl, and M. Ulbrich. Sundance PDE solvers on Cartesian fixed grids in complex and variable geometries. In *Proceedings of the ECCOMAS Thematic Conference CFD & Optimization, Antlya, Turkey, May 23-25, 2011*, May 2011.
9. H. Bijl, A. H. van Zuijlen, and S. Bosscher. Two level algorithms for partitioned fluid-structure interaction computations. In P. Wesseling, E. Oñate, and J. Périaux, editors, *ECCOMAS CFD 2006, European Conference on Computational Fluid Dynamics*. TU Delft, 2006.
10. A. N. Bugrov and S. Smagulov. Fictitious Domain Method for Navier-Stokes Equations. *Mathematical model of fluid flow, Novosibirsk*, pages 79–90, 1978.
11. H.-J. Bungartz, B. Gatzhammer, M. Mehl, and T. Neckel. Partitioned simulation of fluid-structure interaction on Cartesian grids. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid-Structure Interaction – Modelling, Simulation, Optimisation, Part II*, volume 73 of *LNCSE*, pages 255–284. Springer, Berlin, Heidelberg, 2010.
12. H.-J. Bungartz, M. Mehl, and T. Weinzierl. A parallel adaptive Cartesian PDE solver using space–filling curves. In E.W. Nagel, V.W. Walter, and W. Lehner, editors, *Euro-Par 2006, Parallel Processing, 12th International Euro-Par Conference*, volume 4128 of *Lecture Notes in Computer Science*, pages 1064–1074, Berlin Heidelberg, 2006. Springer-Verlag.
13. C. Bursteddea, L.C. Wilcox, and O. Ghattas. p4est: Scalable algorithms for parallel adaptive mesh refinement on forests of octrees. *SIAM Journal on Scientic Computing*, 33(3):1103–1133, 2011.
14. J. Degroote, K.J. Bathe, and J. Vierendeels. Performance of a new partitioned procedure versus a monolithic procedure in fluid-structure interaction. *Comput. Struct.*, 87:793–801, 2009.
15. R. Glowinski, T.W. Pan, T.I. Hesla, D.D. Joseph, and J.Périaux. A Fictitious Domain Approach to the Direct Numerical Simulation of Incompressible Viscous Flow past Moving Rigid Bodies: Application to Particulate Flow. *Journal of Computational Physics*, 169:363–426, 2001.
16. M. Griebel, Th. Dornseifer, and T. Neunhoeffer. *Numerical Simulation in Fluid Dynamics, a Practical Introduction*. SIAM, 1998.
17. M. Griebel and G. Zumbusch. Hash based adaptive parallel multilevel methods with space-filling curves. In H. Rollnik and D. Wolf, editors, *NIC Symposium 2001*, volume 9 of *NIC Series, ISBN 3-00-009055-X*, pages 479–492, Germany, 2002. Forschungszentrum Jülich.
18. Ph. Guillaume and M. Masmoudi. Computation of high order derivatives in optimal shape design. *Numer. Math.*, 67(2):231–250, 1994.
19. F. H. Harlow and J. E. Welch. Numerical calculation of time-dependent viscous incompressible flow of fluid with a free surface. *Physics of Fluids*, pages 2182–2189, 1965.

20. M. Heroux, R. Bartlett, V. Howle, R. Hoekstra, J. Hu, T. Kolda, R. Lehoucq, K. Long, R. Pawlowski, E. Phipps, A. Salinger, H. Thornquist, R. Tuminaro, J. Willenbring, and A. Williams. An Overview of Trilinos. *SAND REPORT*, 2003. http://trilinos.sandia.gov/TrilinosOverview.pdf.
21. M. Hinze, R. Pinnau, M. Ulbrich, and S. Ulbrich. *Optimization with PDE constraints*, volume 23 of *Mathematical Modelling: Theory and Applications*. Springer, New York, 2009.
22. J. Hron and S. Turek. Proposal for numerical benchmarking of fluid-structure interaction between elastic object and laminar incompressible flow. In H.-J. Bungartz and M. Schäfer, editors, *Fluid-Structure Interaction*, number 53 in Lecture Notes in Computational Science and Engineering, pages 371–385. Springer-Verlag, 2006.
23. J. Hron and S. Turek. Numerical benchmarking of fluid-structure interaction between elastic objects and laminar incompressible flow. In H.-J. Bungartz, M. Mehl, and M. Schäfer, editors, *Fluid-Structure Interaction – Modelling, Simulation, Optimisation, Part II*, volume 73 of *LNCSE*. Springer, Berlin, Heidelberg, 2010.
24. U. Kuettler and W.A. Wall. Fixed-point fluid-structure interaction solvers with dynamic relaxation. *Computational Mechanics*, 43(1):61–72, 2008.
25. V.R. Yarberry L.A. Schoof. Exodus II: A Finite Element Data Model. *Sandia Report*, SAND92-2137, 1994.
26. R.J. LeVeque. Cartesian grid methods for flow in irregular regions. In K.W. Morton and M.J. Baines, editors, *Num. Meth. Fl. Dyn. III*, pages 375–382. Clarendon Press, 1988.
27. K. Long. Sundance 2.0 Tutorial, 2004. http://prod.sandia.gov/techlib/access-control.cgi/2004/044793.pdf.
28. K. R. Long, R. C. Kirby, and B. van Bloemen Waanders. Unified embedded parallel finite element computations via software-based Frechet differentiation. *SIAM J. Scientific Computing*, 32(6):3323–3351, 2010.
29. MathWorks. MATLAB The Language of Technical Computing. http://www.mathworks.com/products/matlab/index.html; last visted: September 2012.
30. R. Mittal and G. Iaccarino. Immersed boundary methods. *Annu. Rev. Fluid Mech.*, 37:239–261, 2005.
31. N. Moës, J. Dolbow, and T. Belytschko. A finite element method for crack growth without remeshing. *International Journal for Numerical Methods in Engineering*, 46:131150, 1999.
32. B. Mohammadi and O. Pironneau. *Applied Shape Optimization for Fluids*. Oxford University Press, 2001.
33. F. Murat and J. Simon. Etudes de probl`mes d'optimal design. *Lecture Notes in Computer Science*, 41:54–62, 1976.
34. J. Nitsche. Über ein Variationsprinzip zur Lösung von Dirichlet-Problemen bei Verwendung von Teilräumen, die keinen Randbedingungen unterworfen sind. *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*, 36:9–15, 1971. 10.1007/BF02995904.
35. W. F. Noh and P. Woodward. SLIC (Simple Line Interface Calculation). In A. I. van de Vooren and P. J. Zandbergen, editors, *Proceedings of 5th International Conference of Fluid Dynamics*, Lecture Notes in Physics, pages 330–340. 1976.
36. S. Osher and J. A. Sethian. Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations. *J. Comput. Phys.*, pages 12–49, 1988.
37. C. S. Peskin. Numerical analysis of blood flow in the heart. *J. Comput. Phys.*, pages 220–252, 1977.
38. J.E. Pilliod and E.G. Puckett. Second-order accurate volume-of-fluid algorithms for tracking material interfaces. *Journal of Computational Physics*, pages 465–502, 2004.
39. E. Rank, S. Kollmannsberger, Ch. Sorger, and A. Düster. Shell finite cell method: A high order fictitious domain approach for thin-walled structures. *Computer Methods in Applied Mechanics and Engineering*, 200(45/46):3200 – 3209, 2011.
40. M. Schäfer and S. Turek. Benchmark computations of laminar flow around a cylinder. In *Flow simulation with high-performance computers 2*, volume 52 of *Notes on numerical fluid mechanics*, pages 547–566. Vieweg, 1996.

41. W.J. Schroeder, K.M. Martin, and W.E. Lorensen. The Design and Implementation of an Object-Oriented Toolkit For 3D Graphics And Visualization. In *Proceedings of the 7th Conference on Visualization '96*, VIS '96, pages 93–ff. IEEE Computer Society Press, 1996.
42. J. Sokolowski and J.-P. Zolésio. *Introduction to Shape Optimization: Shape sensitivity analysis*. Springer, Berlin, 1992.
43. T. E. Tezduyar, M. Behr, S. Mittal, and A.A. Johnson. Computation of unsteady incompressible flows with the stabilized finite element methods: space-time formulations, iterative strategies and massively parallel implementations. *Asme Pressure Vessels Piping Div Publ PVP*, 246:7–24, 1992.
44. A. Wächter and L. T. Biegler. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical Programming*, pages 25–57, 2006.
45. T. Weinzierl and M. Mehl. Peano – a traversal and storage scheme for octree-like adaptive cartesian multiscale grids. *SIAM Journal on Scientific Computing*, 33(5):2732–2760, 2011.
46. S. Yigit, M. Heck, D. C. Sternel, and M. Schäfer. Efficiency of fluid-structure interaction simulations with adaptive underrelaxation and multigrid acceleration. *Int. J. Multiphysics*, 1:85–99, 2007.
47. J.-P. Zolésio and M.C. Delfour. *Shapes and Geometries: Analysis, Differential Calculus and Optimization*. SIAM, 2001.